VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Software jako služba

Software as a Service

2020 Radek Novák

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Radek Novák**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Software jako služba
Software as a Service

Jazyk vypracování: čeština

Zásady pro vypracování:

Aktuálním trendem je nabízet software jako online službu, mluvíme o tzv. SaaS – Software as a Service. Tento přístup přináší některé problémy, zejména jak vhodným způsobem zabezpečit a spravovat data. Cílem práce je nastudovat možnosti SaaS, zejména se zaměřením na bezpečnost dat a navrhnout a naimplementovat ukázkovou aplikaci.

Úkoly:
1. Nastudovat možnosti SaaS a porovnat možnosti architektur Single-tentant vs Multi-tenant.
2. Zvolit vhodnou architekturu, navrhnout a naimplementovat funkcionalitu ukázkové aplikace.
3. Implementovat webové rozhraní pro správu SaaS.
4. Optimalizovat datovou vrstvu z pohledu výkonu, zabezpečení a izolace dat.
5. Implementovat jednoduché rozhraní pro uživatele (pronajímatele), tak aby bylo možné aplikaci otestovat.
6. Vytvořit online dokumentaci a porovnat implementované řešení s obdobnými aplikacemi.

Seznam doporučené odborné literatury:

[1] FREEMAN, A., Pro ASP.NET Core MVC. Apress., 2016
[2] Cabrera, James. Modular Design Frameworks : a Projects-based Guide for UI/UX Designers. Berkeley, CA: Apress, 2017
[3] Molina, Hector, Jeffrey D. Ullman, and Jennifer Widom. Database systems : the complete book.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Lukáš Zátopek**

Datum zadání: 01.09.2019
Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: *9. května 2020*

Radek Novák

Rád bych poděkoval Ing. Lukášovi Zátopkovi za odbornou pomoc a konzultaci při vytváření této bakalářské práce.

Abstrakt

Cílem této bakalářské práce je seznámit se s modelem Software jako služba a implementovat informační systém využívající tento model. Zvolený model byl implementován jako systém pro správu úkolů a projektů. Teoretická část práce se zabývá tématem Software jako služba a je provedena analýza možností architektury. Praktická část práce popisuje implementovaný informační systém demonstrující aplikaci modelu Software jako služba, kde je také provedena datová a funkční analýza. Závěr práce je věnován porovnání s již existujícími systémy s podobnou funkcionalitou, implementaci konkrétních funkcí v systému a grafickému rozhraní.

Klíčová slova: Informační systém, Software jako služba, multi-tenant architektura, single-tenant architektura, SaaS, ASP.NET Core

Abstract

The goal of this bachelor thesis is to become acquainted with model Software as a Service and to implement an information system using this model. Model was implemented as task and project management system. The theoretical part delas with topic Software as a Service and contains analysis of architecture options. The practical part of the thesis contains description of the implemented system demonstrating model Software as a Service followed by data and function analysis. The end of thesis is devoted to comparison with existing systems with similar functionality, to system development including chosen functions and to user interface.

Keywords: Information system, Software as a Service, multi-tenant architecture, single-tenant architecture, SaaS, ASP.NET Core

Obsah

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronous JavaScript And Extensible Markup Language
CSS	–	Cascading Style Sheets
ERD	–	Entity Relationship Diagram
HTTP	–	Hypertext Transfer Protocol
HW	–	Hardware
IaaS	–	Infrastructure as a service
IIS	–	Internet Information Services
IS	–	Informační systém
JS	–	JavaScript
PaaS	–	Platform as a service
SaaS	–	Software as a service
SQL	–	Structured Query Language
SW	–	Software
URI	–	Uniform Resource Identifier
URL	–	Uniform Resource Locator

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Moderní doba si žádá moderní technologie a mnoho firem i běžných uživatelů mění způsob užívání informačních technologií směrem ke cloud computingu. Nejedná se o koupi software, nýbrž jde o pronájem služby, která zprostředkovává jak samotný software, tak hardware jím využívaný. Z toho také plyne pojem Software jako služba, dále již jen jako SaaS. Díky tomuto cloudovému řešení lze dosáhnout maximalizaci výkonu a pohodlí ze strany uživatele. Dochází také k minimalizaci ceny, kdy klient nepotřebuje výkonný hardware, nýbrž všechen výkon využívaný thin-client systémy je soustředěn na cloud. V současnosti se obrovské množství vývojářských společností vydává cestou pronájmu aplikací a prostředků. Jmenovat lze například Microsoft s Office 365[1] nebo také Google s Google Apps[2].

Samotná bakalářská práce je rozdělena na teoretickou a praktickou část. Teoretická část je zaměřená na model SaaS a věnuje se možnostem multi-tenant a single-tenant architektury, v rámci kterých může být model implementován. Praktická část práce popisuje implementovaný informační systém demonstrující aplikaci modelu SaaS. Tato část práce se rovněž zaobírá popisem datové a funkční analýzy. Datová analýza je zaměřena na zvolenou architekturu a zaobírá se problémem izolace, ochrany a výkonu dat. Následuje funkční analýza, která popisuje jádro funkcionality systému. Následující část se zabývá porovnáním již zavedených systémů pro správu úkolů a projektů. Závěr práce je věnován konkrétním implementacím nosných funkcí a ukázce vytvořeného grafického rozhraní pro správu tohoto systému.

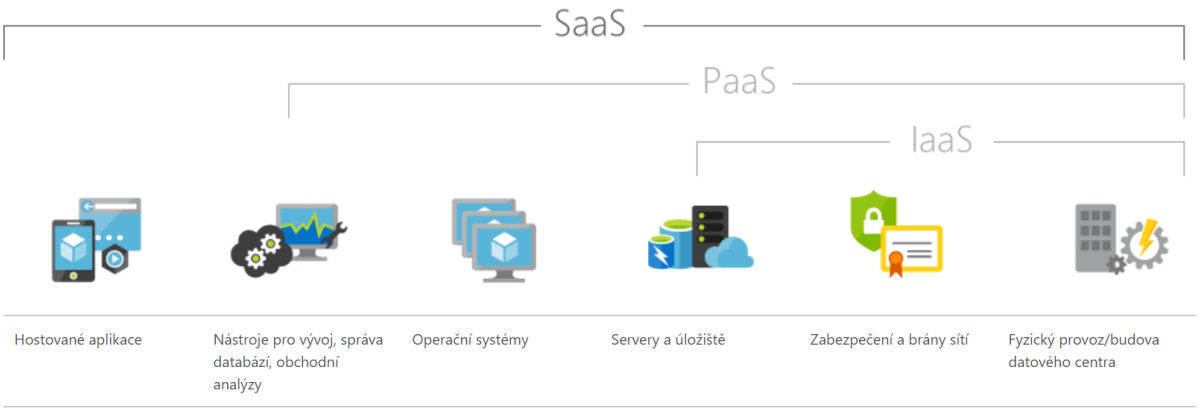
Cílem této práce je vytvořit informační systém, na kterém budou aplikovány nabyté poznatky o SaaS architektuře, s cílem izolovat data mezi jednotlivými uživateli a s dostatečným výkonem v rámci dotazování nad daty. Výstupem bude informační systém pro správu úkolů a projektů, který usnadňuje a zpřehledňuje rozdělení práce v týmu. Téma informačního systému bylo vhodně zvolené pro otestování modelu SaaS.

2 Model software jako služba

2.1 Úvod

SaaS jako pojem nabývá na čím dál větší intenzitě v moderní době, kdy se velká část IT světa orientuje na užívání prostředků cloudu. Model SaaS funguje na bázi předplatného, kdy si klient na základě plateb pronajímá službu, která poskytuje kompletní řešení aplikace. To znamená, že poskytovatel poskytuje samotnou aplikaci, infrastrukturu, na které aplikace běží, a také veškerou podporu zahrnující zálohy dat, zabezpečení, nasazení nových verzí aplikace apod. Díky tomu, že řešení jako celek běží na cloudu na straně poskytovatele, uživatelům stačí pro užívání aplikace pouze připojení k internetu a webový prohlížeč. Díky SaaS se výrazně omezí pořizovací náklady pro podniky, kterým odpadá povinnost pořízení infrastruktury a také trvalé jednorázové licence. Oproti klasickým aplikacím se poznatelně zmenší čas, který je potřebný k nasazení aplikace. U aplikací využívajících modelu SaaS mnohdy stačí pouhá registrace a několik prvotních nastavení. [3]

Jako příklad využití SaaS můžeme uvést aplikace, které nevyžadují složitější propojení s interními podnikovými systémy, nebo také aplikace pro jednotlivé uživatele provozované na cloudu. V obou případech platí, že pronajímatel si dané aplikace předplácí. Nejedná se o klasickou platbu za licenci, nýbrž poplatek za jeho používání. SaaS je lukrativní také pro podniky, kde je vyžadován profesionální SW a nejsou k dispozici jednorázové prostředky pro vývoj.



Obrázek 1: SaaS model [3]

Obrázek 1 demonstruje rozsah jednotlivých obchodních modelů poskytovaných v cloudovém prostředí. Nejnižší cloudový model IaaS poskytuje pouze výpočetní prostředky jako např. servery či databáze a veškerý SW (včetně operačního systému) si musí zákazník spravovat sám. Model, který poskytuje celkové prostředí, kde můžeme provozovat vlastní SW se označuje jako PaaS. Model SaaS nabízí kompletní řešení pro jeden konkrétní SW, který si zákazník pronajímá. [4]

2.2 Klady a zápory SaaS

Abychom porozuměli vzrůstající popularitě SaaS, je třeba si určit kladné a záporné faktory na straně poskytovatele i nájemce.

Nejdříve se zaměříme na finanční stránku, která je výhodná pro obě strany. Díky modelu, kde je platba řešena předplatným na určené období, je zaručen pravidelný příjem pro poskytovatele služeb. Na druhou stranu u nájemce dochází k rozložení nákladů za licenci, kterou nemusí platit jako jednorázovou, nýbrž je platba rozložena. Další nespornou výhodou na straně nájemce (především u menších společností a startupů) je redukce ceny HW, kde není třeba investovat do serverů a pro uživatele je možno pořídit slabší koncové stanice, protože většina výkonu je obstarána poskytovatelem. Většinou je také nájemci poskytnuto více možností služeb, takže si může vybrat tu, která nejvíce odpovídá business modelu dané společnosti a nemusí platit za kapacity aplikace, které nevyužije. [5]

Dalším faktorem ovlivňujícím rozhodnutí o nasazení SaaS je zabezpečení. Zabezpečit data není jednoduché a pro vysokou bezpečnost je potřeba tým profesionálů, který často menší firmy nemají k dispozici a implementují tak méně bezpečnější řešení, než poskytovatelé cloudových služeb. Zde se také dostáváme k potenciálnímu velkému problému, kdy nájemci nemají svá data umístěna lokálně, ale na cloudu. Nájemce nikdy neví, zdali poskytovatel služeb nebude provádět nějaké operace nad jeho daty bez souhlasu. Je zde potřeba stoprocentní důvěra ze strany nájemce a musí se spoléhat na poskytovatele služeb, že nedojde k úniku nebo poškození dat. [6]

Obrovská výhoda nastává při práci v týmu, kde se zjednodušuje týmová flexibilita a spolupráce. Díky cloudovému řešení může tým spolupracovat na projektech přes internet v reálném čase. K tomu se také pojí možnost využití více platforem, kdy uživatel může přistupovat ke stejným datům z libovolného počítače či mobilního zařízení. Stačí pouze připojení k internetu. Dále se také nemusíme obávat ztráty dat při havárii počítače nebo jiného koncového zařízení, protože všechna data zůstávají uložena na cloudu. Je zde také odhadována menší chybovost, než v případě lokálního hostování SW. [3, 5]

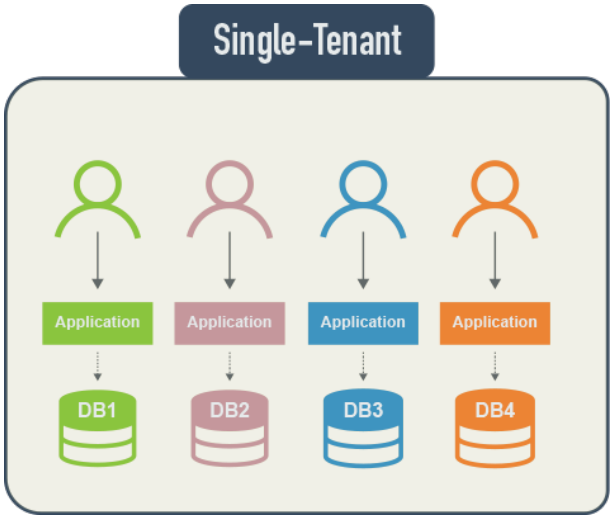
Zvážit také musíme rychlost připojení k internetu, na kterou budou kladeny vyšší nároky s každou aplikací využívající cloud computing.

3 Možnosti architektury

Rozhodnutí je potřeba provést na základě několika otázek – jaké jsou dostupné finanční prostředky? Jaké jsou požadavky na zabezpečení dat? Je potřeba přizpůsobit SW větší mírou pro jednotlivé klienty? Odpovědí na tyto otázky jsou přiblíženy v následujících kapitolách, kde je uveden popis a srovnání základních architektur, které model SaaS nabízí.

3.1 Single-tenant architektura

Single-tenant architektura znamená, že každá instance aplikace a prostředí, ve kterém běží, slouží pouze pro jednoho nájemce (viz obrázek 2). Dochází k oddělení dat i výkonu mezi nájemci, díky čemuž má poskytovatel možnost nabízet přizpůsobení SW i HW potřebám nájemce. Mezi základní vlastnosti single-tenant architektury se řadí spolehlivost, bezpečnost nebo také snadná záloha dat. [7]



Obrázek 2: Single-tenant architektura [8]

3.1.1 Výhody a nevýhody single-tenant architektury

Single-tenant architektura se charakterizuje kompletní izolací dat od ostatních nájemců a také může dosahovat vyššího výkonu. Maximální izolace se často dosahuje pomocí přiřazení jednotlivých pronajímatelů na samostatné servery. Velmi často tak tedy může platit, co jeden pronajímatel, to jeden server. Je ale třeba brát v potaz, že tato realita se odráží na ceně, která je ovlivněna infrastrukturou čítající několikanásobně větší množství serverů. S každým serverem přichází také povinnost jeho správy, což může být ve větších infrastrukturách nákladné ze strany poskytovatele. V tabulkách 1 a 2 jsou uvedeny výhody a nevýhody single-tenant architektury [7]

Tabulka 1: Výhody single-tenant architektury [9, 7]

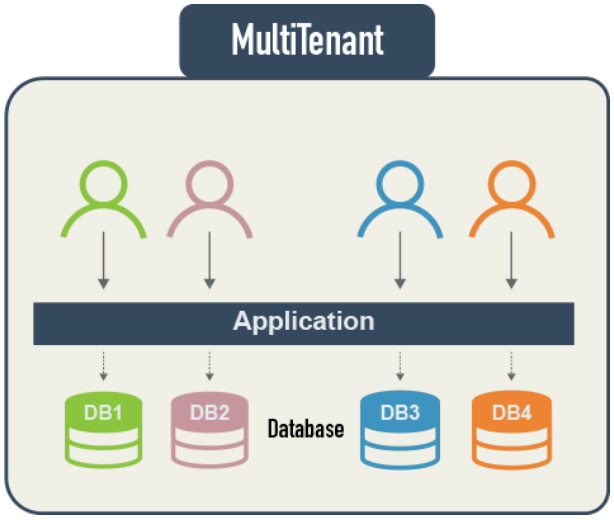
Bezpečnost	Single-tenant architektura nabízí kompletní separaci dat od ostatních nájemců. Zpravidla to bývá tak, že jednomu nájemci je přiřazen právě jeden server. Je tak velice nepravděpodobné, že by docházelo k úniku dat mezi jednotlivými nájemci.
Spolehlivost	Tato architektura je považovaná za spolehlivější, protože každý nájemce má vlastní instanci aplikace na samostatném serveru. Tudíž výpadek, či velká vytíženost prostředků ze strany jednoho nájemce neovlivní ostatní.
Rychlost	Výkon serveru je soustředěn pouze pro jednoho nájemce, tím pádem má nájemce k dispozici veškeré prostředky serveru a rychlost není snížena vytížeností ostatních nájemců.
Obnova dat	Každý nájemce má separátní zálohy dat, nemusí se implementovat složité řešení pro obnovu rámci všech nájemců.
Konfigurace	Single-tenant architektura nabízí konfiguraci SW i HW. Díky samostatné instanci a prostředí, ve kterém aplikace běží, je možnost přizpůsobit aplikaci nebo také škálovat prostředky serveru dle požadavků nájemce. Nájemci je také většinou umožněno spravovat nové aktualizace a vylepšení.

Tabulka 2: Nevýhody single-tenant architektury [10, 9, 7]

Cena	V single-tenant architektuře je mnoho faktorů zvyšující cenu. Začíná to náročnějším vývojem automatizace, dále je potřeba rozsáhlé infrastruktury, náročnější monitoring a údržba, a také větší spotřeba energie. Tyto faktory se poté promítnou i do ceny samotného pronájmu SW.
Nasazení	Nasazení aplikace běžící na single-tenant architektuře je složitější. Je zapotřebí zařídit rozsáhlejší infrastrukturu. Dále je zde náročnější implementace automatizace např. registrace nového nájemce.
Provoz	Díky rozsáhlé infrastruktuře se udržuje a monitoruje velké množství serverů, což zvyšuje náročnost na lidské i technické zdroje.
Efektivita	Kvůli velkému množství instancí a serverů zde nemusí docházet k efektivnímu využití výkonu prostředků. Je zde náročnější přesně přidělit HW prostředky a může docházet k větší nevyužité kapacitě HW.

3.2 Multi-tenant architektura

Multi-tenant architektura definuje, že všichni nájemci využívají sdílené instance aplikace (viz obrázek 3). Tato aplikace běží většinou na jednom serveru, ale v případě většího množství nájemců je využito několika serverů, kde zpravidla bývá potřeba pro správné rozložení zátěže implementovat load balancer. Díky centralizované struktuře dochází ke snížení počátečních i provozních nákladů a jednodušší správě infrastruktury. Na druhou stranu je zde potřeba, aby algoritmy kladly zvětšený důraz na izolaci dat jednotlivých nájemců z důvodu sdílené instance aplikace.[11]



Obrázek 3: Multi-tenant architektura [8]

U multi-tenant architektury se implementace aplikační logiky nijak neliší. Hlavní rozdíl nastává u problému izolace dat a dotazování nad daty, kde je nutno soustředit velkou pozornost. Existují tři hlavní postupy implementace, kterým se budou věnovat následující kapitoly. Tyto tři postupy jsou:

1. **Sdílená databáze, sdílené schéma** – kap. 3.2.2
2. **Sdílená databáze, izolované schéma** – kap. 3.2.3
3. **Izolovaná databáze** – kap. 3.2.4

3.2.1 Výhody a nevýhody multi-tenant architektury

Multi-tenant architektura je dobrá volba pro společnosti, které mají menší požadavky na kompletní izolovanost dat či nemají vyčleněný dostatek financí pro předplácení podobné single-tenant aplikace. I když se multi-tenant prostředí může zdát ideální, stále existují limity a rizika, na které si musíme jak při implementaci, tak při provozu dát pozor. Výhody a nevýhody jsou znázorněny v tabulkách 3 a 4.

Tabulka 3: Výhody multi-tenant architektury [10, 9]

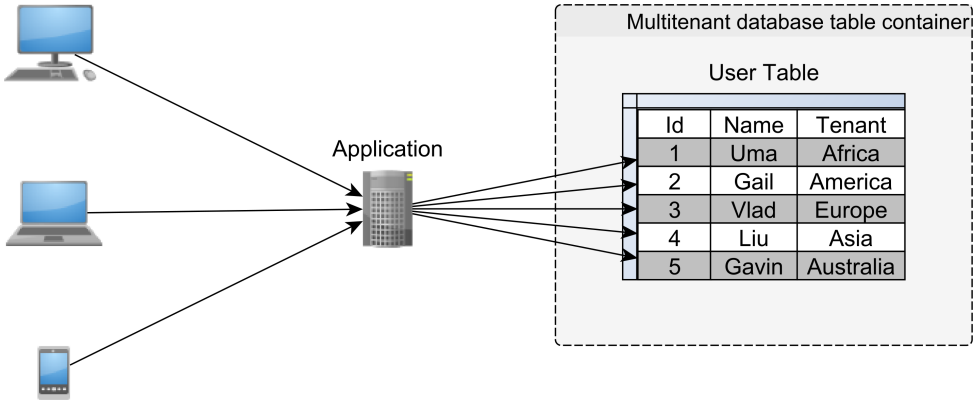
Finance	Cena za jednotlivého nájemce je nižší, protože všechny prostředky jsou sdílené a není vyžadováno rozsáhlé infrastruktury. Díky centralizaci aplikace se ušetří také na monitoringu, údržbě serverů a nasazení nových verzí SW.
Využití prostředků	Multi-tenant architektura zajišťuje efektivnější využití výkonu. Nejsou zde rezervy pro každého nájemce zvlášť a dá se lépe propočítat potřebné prostředky serveru a tím maximalizovat jejich využití.
Automatizace	Implementace automatizace není nikterak náročná. Za zmínku stojí především automatizace registrace nového nájemce.
Údržba	Sdílená instance nabízí jednoduchou správu nových aktualizací a vylepšení. Dále také není zapotřebí se starat o tak rozsáhlou serverovou infrastrukturu.
Ochrana	Infrastruktura aplikace je distribuována v rámci malého množství serverů, tím pádem se centralizuje i bezpečnost a existuje méně cest, jak danou infrastrukturu napadnout.

Tabulka 4: Nevýhody multi-tenant architektury [10, 9]

Bezpečnostní riziko	U multi-tenant architektury je zapotřebí brát v potaz bezpečnost dat mezi jednotlivými nájemci. Aby se docílilo izolace dat mezi nájemci, nastává nutnost implementovat striktní autentizační a přístupové kontroly. V případě, že útočník získá přístup k serveru, jsou ohrožena data všech nájemců.
Udržitelnost	V případě pádu serveru nastává pád pro všechny nájemce najednou. Tato skutečnost nastává také při aktualizacích či vylepšeních aplikace nebo HW, kde je potřeba určitého servisního okna v provozu.
Sdílení prostředků	Při zvyšujícím se počtu nájemců musí poskytovatel přidávat kapacitu systémových prostředků. V případě velkého množství nájemců je potřeba implementace efektivního load balanceru. Při nesprávném rozložení zátěže může dojít k „soutěžení“ o systémové prostředky mezi jednotlivými nájemci.
Záloha a obnova dat	V multi-tenant architektuře je třeba důsledně dbát na zálohy dat, protože když dojde k jejich poškození, je velká šance, že se poškodí data u všech nájemců a mělo by to velké dopady. Záloha a obnova dat většinou vyžaduje složitější algoritmická řešení.

3.2.2 Sdílená databáze, sdílené schéma

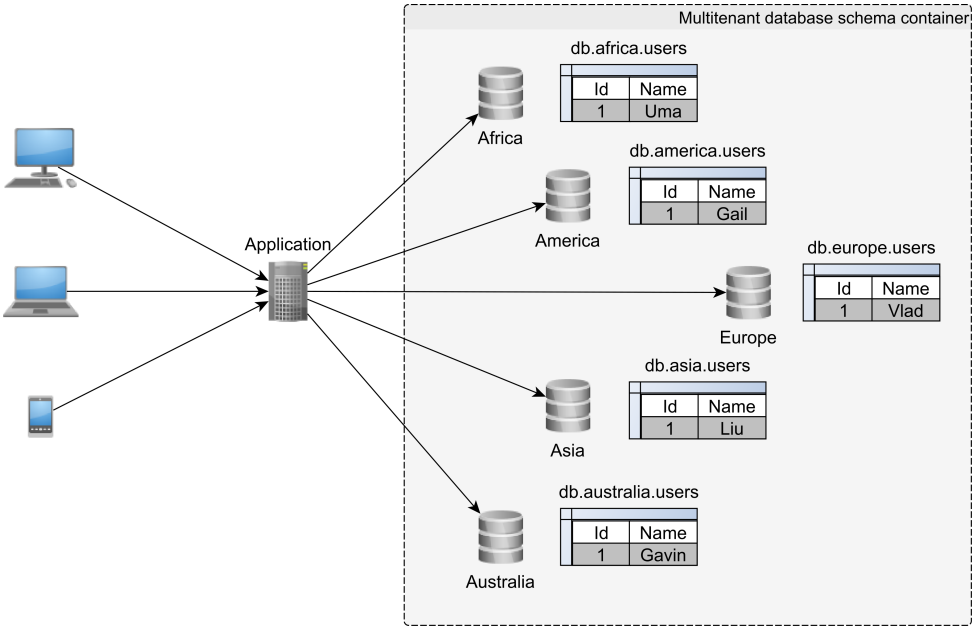
Tento postup je založen na ukládání dat všech nájemců v jedné databázi a v jednom schématu. Oddělení dat jednotlivých nájemců je řešeno cizím klíčem nájemce, který je uveden v každé tabulce. Izolace dat je řešena v aplikační logice, kde je nutno implementovat algoritmy, aby při dotazování na data nedošlo k úniku dat mezi jednotlivými nájemci. Vzhledem k tomu, že databázové tabulky obsahují data všech nájemců, mohou nabývat nemalých rozměrů a může dojít k prodloužení času dotazování na data a k celkovému zpomalení chodu aplikace. Tato realita je zapříčiněna procházením irelevantních dat od ostatních nájemců. [12, 13]



Obrázek 4: Sdílená databáze, sdílené schéma [12]

3.2.3 Sdílená databáze, izolované schéma

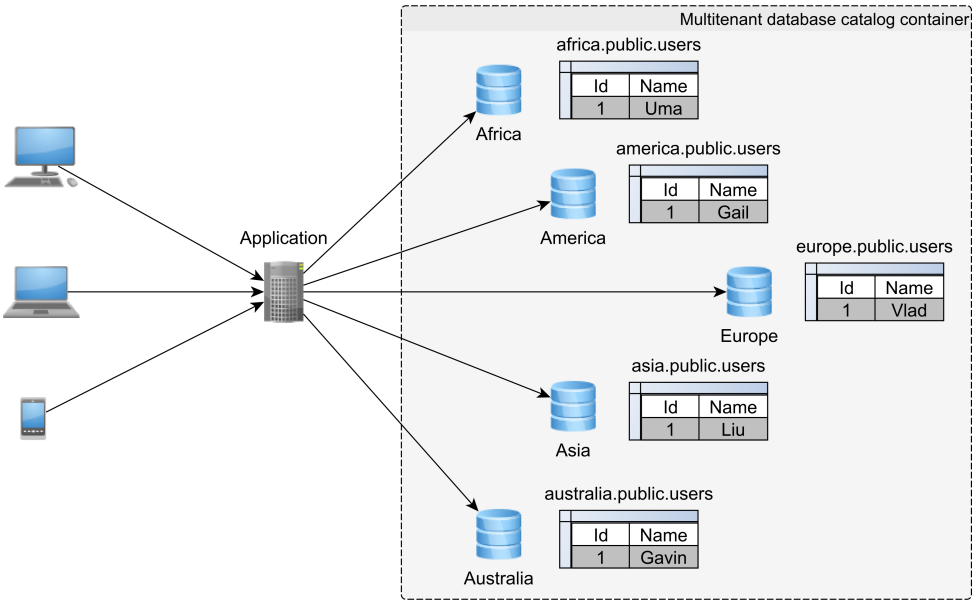
Podobně jako v předchozím řešení je u tohoto postupu sdílena jedna databáze pro celou aplikaci. Avšak pro každého nájemce je vytvořeno v databázi samostatné schéma, kde se ukládají jeho data, která jsou logicky oddělena od ostatních nájemců. Izolace dat je řešena pomocí uživatelských práv a přístupů na straně databáze, kdy je pro každého nájemce vytvořen samostatný uživatel s právy pouze pro vlastní schéma, popřípadě schéma sdílené. Není potřeba implementovat komplexní řešení v aplikační logice, kde se pouze řeší, jakým způsobem se aplikace připojí k databázi a nabízí se více prostoru pro implementaci business logiky. Implementaci postupu sdílené databáze a izolovaného schéma se detailněji věnuje kapitola 6.3, kde je ukázána logika na aplikační i databázové straně. [12, 13]



Obrázek 5: Sdílená databáze, izolované schéma [12]

3.2.4 Izolovaná databáze

V implementačním postupu izolovaná databáze je každému nájemci vytvořena samostatná databáze, kde se ukládají jeho data. Díky tomuto mohou být data oddělena také fyzicky. Podobně jako u předchozího postupu, není potřeba implementovat komplexní řešení v aplikační logice, která se zabývá pouze tím, nad jakou databází se vykoná dotaz na data. Problém může nastat při monitorování nebo záloze dat, kde je třeba implementovat náročnější automatizaci.[12, 13]



Obrázek 6: Izolovaná databáze, izolované schéma [12]

3.3 Srovnání architektur

Rozdíl mezi multi-tenant a single-tenant architekturou je graficky znázorněn na obrázcích 2 a 3, které ilustrují způsob sdílení instance aplikace mezi jednotlivé nájemce. Ze shrnutí výhod a nevýhod obou architektur vyplývá, že multi-tenant architektura je ideální pro menší společnosti, či startupy, které nemají vyčleněné velké množství financí pro pronajmutí obdobné single-tenant aplikace a netrvají na kompletním fyzickém oddělení dat od ostatních nájemců. Tato architektura se stala standardem pro podnikové SaaS prostředí. Oproti single-tenant architektuře je levnější, efektivnější na využití HW prostředků a vyžaduje méně nákladů na údržbu ze strany pronajímatele. Naproti tomu single-tenant architektura, která se zdá být lepším řešením pro korporátní prostředí, nabízí spolehlivější chod aplikace s výkonem neovlivněným vytížením ostatních nájemců. Dále poskytuje lepší izolaci dat, která jsou fyzicky oddělena od ostatních nájemců, a tím tak chrání data před úniky. Tato architektura, stavěná na rozsáhlé infrastruktuře, je nákladnější na straně pronajímatele i zákazníka. [11, 10, 7]

Daná pozitiva či negativa u některých produktů nemusí hrát žádnou roli, proto rozhodnutí nejlepšího řešení není možné udělat mezi single-tenant a multi-tenant architekturou, ale vždy je třeba zvolit produkt, který vyhovuje požadavkům zákazníka.

4 Systém pro správu úkolů a projektů

4.1 Úvod

Součástí této bakalářské práce je implementace IS, který demonstruje model SaaS. Zvoleným tématem tohoto IS je správa úkolů a projektů. Při implementaci byla zvolena multi-tenant architektura. Tato volba byla provedena v zásadě kvůli snadnější možnosti testování a daleko menší infrastruktuře serverů. Dalším faktorem, který vedl k této volbě, byla daleko snazší správa systémových prostředků a nájemců.

Implementovaný systém nabízí usnadnění rozdělení práce v týmu pomocí přiřazování úkolů, které jsou dále součástí vytvořených projektů. Systém využívá různé uživatelské role (např. manažer, administrátor), které jsou popsány v kapitole 4.4.1.1. Systémem jsou nabízeny tři varianty poskytovaných služeb, lišící se nabídkou dostupných funkcionalit a omezením kapacit. Nabídka je stupňována od funkcí základních až po funkce prémiové. Za služby je potřeba platit měsíční poplatky vyjma služby BASIC, která je zdarma. Služba BASIC nabízí omezenou funkčnost systémů – dovoluje pouze jeden projekt, pět uživatelů na doménu a pět komentářů na jeden úkol. Vylepšená verze PRO je nabízená společně s veškerou funkcionalitou, nicméně je omezena stanovenými limity – pro doménu dovoluje pět projektů, deset uživatelů a maximálně deset komentářů na úkol. Služba Business je nejvyšší variantou služeb, je nabízená s veškerou funkcionalitou a bez vymezujících limitů.

Projekty jsou základní stavební jednotkou systémů. Každý projekt se může skládat z vícero úkolů, přičemž součástí projektu jsou uživatelé, kterým jsou jednotlivé úkoly přiřazeny. Nicméně pro přiřazení uživatele k úkolu musí být tento uživatel součástí stejného projektu, jako uživatel, který úkol zadal. Přizvat nové uživatele do projektu má pravomoc pouze vedoucí projektu, který mimo jiné může také jmenovat další vedoucí projektu. V rámci jednoho projektu je možné k úkolu přiřadit jednoho nebo vícero uživatelů. Úkol mohou komentovat všichni, kteří jsou k úkolu přiřazeni. Úkol může nabývat dvou stavů, nesplněný/splněný. Do stavu splněný přejde pouze v okamžiku, kdy jej označí za splněný všichni uživatelé, kterým byl úkol přiřazen. Systém také disponuje mechanismem upozornění, který uživatele upozorňuje na nové úkoly nebo také na blížící se termín splnění úkolu.

4.2 Použité technologie

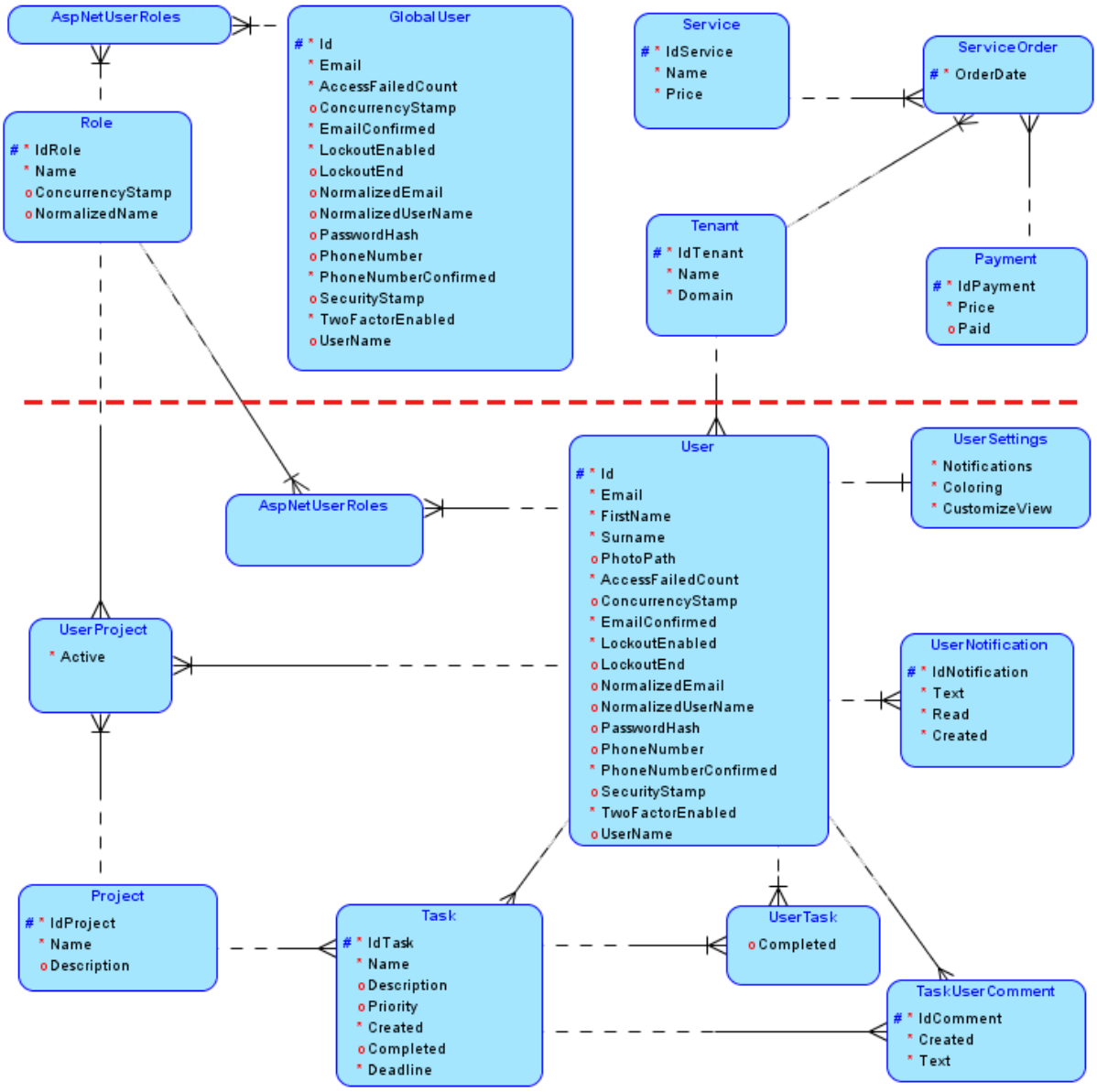
Implementovaný systém využívá běžně používaný architektonický vzor klient-server. Na straně serveru je využitý webový framework ASP.NET Core [14]. Pro dynamičnost prezentační vrstvy je využíván JS [15] a část dotazů je řešena asynchronním voláním metod pomocí techniky AJAX [16]. K dosažení požadovaného designu je využitý framework Bootstrap [17] doplněný vlastním CSS [18] kódem.

Jako databázový server je použit SQL server 2019 [19], kde se také nachází část logiky ve formě uložených procedur. Zbytek aplikační logiky sloužící k přístupu k datům je řešen pomocí Entity Framework Core [20].

V případě nasazení do produkčního prostředí by byl použit webový server IIS [21], který by běžel na Windows Serveru [22]. Pro vývoj a testování je použit webový server IIS Expres [23].

4.3 Datová analýza

4.3.1 ERD modely



Obrázek 7: Logický model databáze

Na obrázku 7 je znázorněn logický model databáze rozdělený červenou přerušovanou čarou. Tato čára značí rozdělení sdílených tabulek (nad čarou), kde jsou uložena sdílená data pro všechny nájemce. Zatímco tabulky pod čarou jsou vytvořeny pro každého nájemce v separátním schématu, kdy každý nájemce má k dispozici pouze svá data. Relační model databáze se pro větší přehlednost nachází v příloze jako obrázek 11.

4.3.2 Izolace dat

Aplikace využívá implementačního postupu sdílené databáze a izolovaného schématu. Tento postup byl zvolen pro logickou separaci dat jednotlivých nájemců a také pro omezení procházení irelevantních dat. Izolace dat je řešena pomocí separátních schémat, kdy k danému schématu nájemce má na straně databáze přístup pouze jeden uživatelský účet, který je jedinečný pro každého nájemce. Při registraci nového nájemce tedy dojde k vytvoření samostatného schématu s potřebnými daty a založení nového uživatelského účtu, který může přistupovat pouze k datům nově vytvořeného schématu. Ostatní data v jiných nesdílených schématech jsou mu skryta a nemá k nim přístup. Část aplikační logiky je řešena pomocí uložených procedur, které vyžadují větší míru bezpečnosti. Z těchto důvodů byl vytvořen uživatelský účet, který na SQL serveru disponuje rolí *securityadmin* a v rámci databáze má roli *db_owner*. Tyto role jsou potřebné pro založení nového loginu, uživatele a schématu. V aplikační logice je izolace docíleno pomocí dynamického vytváření připojovacího řetězce na databázi, kde se zajistí připojení na databázový server uživatelským účtem konkrétního nájemce.

4.3.3 Optimalizace a výkon

V multi-tenant architektuře mohou být dotazy na data pomalé. Toto nastává hlavně při implementačním postupu sdílená databáze, sdílené schéma, kde dochází k procházení dat ostatních nájemců. S rostoucím počtem nájemců dochází ke zvětšování počtu záznamů v tabulkách. Při uvažování situace, kdy by implementovaná aplikace měla 1001 nájemců a každý by měl vytvořených 1000 úkolů, by se při dotazu na data úkolů procházelo milion irelevantních záznamů, což by mohlo vést k navýšení času vykonání dotazu. Největší problém nastává s textovým vyhledáváním, kde se doba zpracování dotazu, v tabulkách obsahujících miliony záznamů, může vyšplhat až na několik vteřin. Optimalizování tohoto problému zajistil implementační postup sdílená databáze, izolované schéma. Zde jsou data logicky oddělena a veškeré operace prováděné nájemcem jsou prováděny pouze nad jeho schématem, kde jsou uložena data pouze jednoho nájemce. Nedochází tedy k procházení irelevantních dat a jedná se o dostatečnou optimalizaci pro hladký běh systému. Pro optimalizaci a zrychlení dotazů na data byly vybrány nejčastěji používané dotazy, u kterých byly vytvořeny indexy.

4.4 Funkční analýza

4.4.1 Uživatelé a jejich role

4.4.1.1 Uživatelské role v rámci aplikace

- **Administrátor**
  Administrátor je speciální role, která umožňuje přístup do prostředí správy SaaS, kde může jednotlivým nájemcům zvýšit/snížit předplacenou službu, přidělit dny zdarma, popřípadě zablokovat doménu nájemce, dokud neproběhne platba.

- **Nájemce**
  Uživatel disponující touto rolí může zvýšit či snížit stupeň využívané služby pro všechny uživatele pod doménou daného nájemce. Dále může spravovat veškeré projekty a uživatelské role v rámci aplikace. Uživatel vlastnící tuto roli má také roli manažera a uživatele.

- **Manažer**
  Role manažera dovoluje uživatelům zakládat nové projekty, kde se po vytvoření stává vedoucím projektu. Uživatel disponující touto rolí má také roli uživatele.

- **Uživatel**
  Uživatel je osoba, která se zaregistruje do domény nájemce a nejsou ji přidělena žádná další privilegia. Po registraci může uživatel pouze upravovat profil. Další práva se mu naskytnou po přiřazení do projektu manažerem nebo nájemcem. Zde se pak vztahují role v rámci projektu popsané v následující kapitole.

4.4.1.2 Uživatelské role v rámci projektu

- **Vedoucí projektu**
  Roli vedoucího projektu získá uživatel při vytvoření projektu nebo když je mu přidělena jiným vedoucím projektu. Tato role dává právo upravovat a smazat projekt, pozvat do projektu nové uživatele a měnit role uživatelů v rámci projektu.

- **Projektový uživatel**
  Jedná se o základní roli uživatele v projektu. Role umožňuje prohlížet projekt, zakládat a přidělovat v něm jednotlivé úkoly. Tyto úkoly, které přidělil, může následně editovat či smazat.

4.4.2 Popis funkcí

4.4.2.1 Funkce v administrátorském prostředí

1. **Vypsat seznam nájemců**
   Umožní vypsat seznam všech nájemců využívajících aplikaci s parametry využívaných služeb.

2. **Změnit předplacenou službu**
   Změní nájemci předplacenou službu a přidá uvedený počet dní využívání služby zdarma.

3. **Přidat dny zadarmo**
   Funkce zkontroluje, zda nájemce využívá placenou službu, pokud ano, přidá zadaný počet dnů, kdy nájemce využívá placené služby zdarma.

4. **Zakázat předplacenou službu**
   Funkce nastaví nájemci poslední provedenou platbu za propadlou a znemožní nájemci používat aplikaci, dokud neproběhne další platba.

5. **Přihlásit**
   Umožňuje uživateli přihlásit se do systému pomocí uživatelského jména a hesla v rámci validačních pravidel.

6. **Odhlásit**
   Umožní odhlášení uživatele ze systému.

4.4.2.2 Uživatelské funkce

1. **Registrovat**
   Funkcionalita umožňující registraci do domény a registraci nového nájemce. Diagram aktivit a scénáře případů užití pro tuto funkci se nachází v kapitole 4.4.3.

2. **Přihlásit**
   Umožňuje uživateli přihlásit se do systému pomocí uživatelského jména a hesla v rámci validačních pravidel.

3. **Odhlásit**
   Umožní odhlášení uživatele ze systému.

4. **Nahrát profilový obrázek**
   Nahraje uživatelskou fotku na server v rámci specifikované URI adresy.

5. **Změnit heslo**
   Změní uživateli heslo pro přístup do systému.

6. **Změnit uživatelské nastavení**
   Umožňuje změnit hodnoty uživatelského nastavení aplikace, které jsou k dispozici pouze pro placené služby. Nastavuje zobrazování upozornění, zabarvení úkolů a umožňuje skrývání vybraných projektů.

7. **Validovat zaplacenou službu**
   Validuje poslední provedenou platbu nájemce. V případě nevalidní platby znemožní uživateli využití aplikace do doby, než je nájemcem uskutečněná platba za využívanou službu.

4.4.2.3 Funkce využívané nájemcem

1. **Přidat/odebrat uživateli roli manažera**
   Přiřadí/odebere uživateli roli manažera. Uživatel je poté o provedené akci upozorněn.

2. **Zaplatit/změnit předplacenou službu**
   Funkce vytvoří novou platbu a objednávku služby. Pokud se jedná o změnu služby, platba je evidována na 30 dnů. V případě prodloužení platby se přičte počet dní, které zbývaly do vypršení platby, k novým třiceti dnům.

4.4.2.4 Funkce projektů

1. **Přidat nový projekt**
   Vytvoří nový projekt a přiřadí zakládajícího uživatele do role vedoucí projektu.

2. **Upravit projekt**
   Aktualizace parametrů projektu.

3. **Odstranit projekt**
   Odstranění projektu ze systému včetně všech přidružených závislostí.

4. **Zobrazit projekty**
   Funkce zobrazí veškeré projekty uživatele. V případě, že je uživatel vedoucí projektu, zobrazí také možnosti pro jeho úpravu či smazání.

5. **Zobrazit projektové žádosti**
   Umožní zobrazit seznam žádostí o vstup do projektů.

6. **Pozvat uživatele do projektu**
   Funkcionalita zašle uživateli žádost o vstup do projektu.

7. **Potvrdit/zamítnout projektovou žádost**
   Funkce přiřadí/odebere uživatele do/z projektu.

8. **Změnit roli uživatele v projektu**
   Funkce mění roli zvoleného uživatele v rámci projektu a upozorní ovlivněného uživatele o provedené změně. Může být vykonána pouze jeho vedoucím.

9. **Změnit viditelnost projektu**
   Změní stav projektu - aktivní/neaktivní. Úkoly neaktivních projektů nejsou viditelné v seznamu úkolů.

4.4.2.5 Funkce úkolů

1. **Přidat nový úkol**
   Vytvoří nový úkol a upozorní uživatele, kterým byl úkol přiřazen.

2. **Upravit úkol**
   Aktualizuje parametry úkolu. Následně upozorní případné uživatele, kterým byl úkol nově přiřazen, a uživatele, kteří byli z úkolu odebráni. Tuto funkci může vykonat pouze uživatel, který úkol zadal.

3. **Odstranit úkol**
   Funkce odstraní úkol včetně všech přidružených závislostí. Funkci může provést pouze uživatel, který úkol zadal.

4. **Přidat komentář**
   Funkce zkontroluje, zda je možno přidat další komentář dle úrovně předplacené služby, přidá komentář a upozorní uživatele, který úkol zadal.

5. **Odstranit komentář**
   Odstraní komentář uživatele.

6. **Zobrazit úkoly**
   Funkce zobrazí veškeré úkoly, které jsou uživateli přiděleny nebo je zadal. Úkoly jsou rozděleny do skupin - nadcházející, propadlé a splněné. Funkce se při zobrazování úkolu řídí uživatelským nastavením, zda je vyžadováno zobrazit jen vybrané projekty, či zda se zbývající čas úkolů zvýrazní barvami.

7. **Zobrazit detail úkolu**
   Zobrazí všechny parametry úkolu.

8. **Hledat úkol**
   Zobrazí úkoly, kterých je uživatel součástí, filtrované dle jména úkolu zadaného uživatelem.

9. **Splnit úkol**
   Označí úkol za splněný pro daného uživatele. Když je úkol vykonán všemi přiřazenými uživateli, je označen za splněný. Následně upozorní uživatele, který úkol zadal.

4.4.2.6 Funkce upozornění

1. **Zobrazit počet upozornění**
   V případě, že má uživatel zapnutou možnost upozornění, zobrazí počet nepřečtených upozornění. Pokud uživatel nemá žádné upozornění nebo má tuto možnost vypnutou, zobrazí počet žádostí ke vstupu do projektů.

2. **Zobrazit přečtené/nepřečtené**
   Kontroluje, zda uživatel využívá možnost upozornění. Pokud ano, zobrazí vybrané upozornění.

3. **Přečíst upozornění**
   Označí vybrané upozornění za přečtené.

4. **Označit všechny upozornění jako přečtené**
   Funkce přenastaví všechny nepřečtené upozornění uživatele na přečtené.

5. **Upozornit uživatele**
   Přidá nové upozornění dle vykonané aktivity a uživatele, kterého se týká.

4.4.3 Případ užití registrace

- **Scénář**
  Funkce registrace obsahuje dva možné průběhy scénáře (registrace nového nájemce, registrace do již registrované domény), které jsou znázorněny v tabulkách 5 a 6.

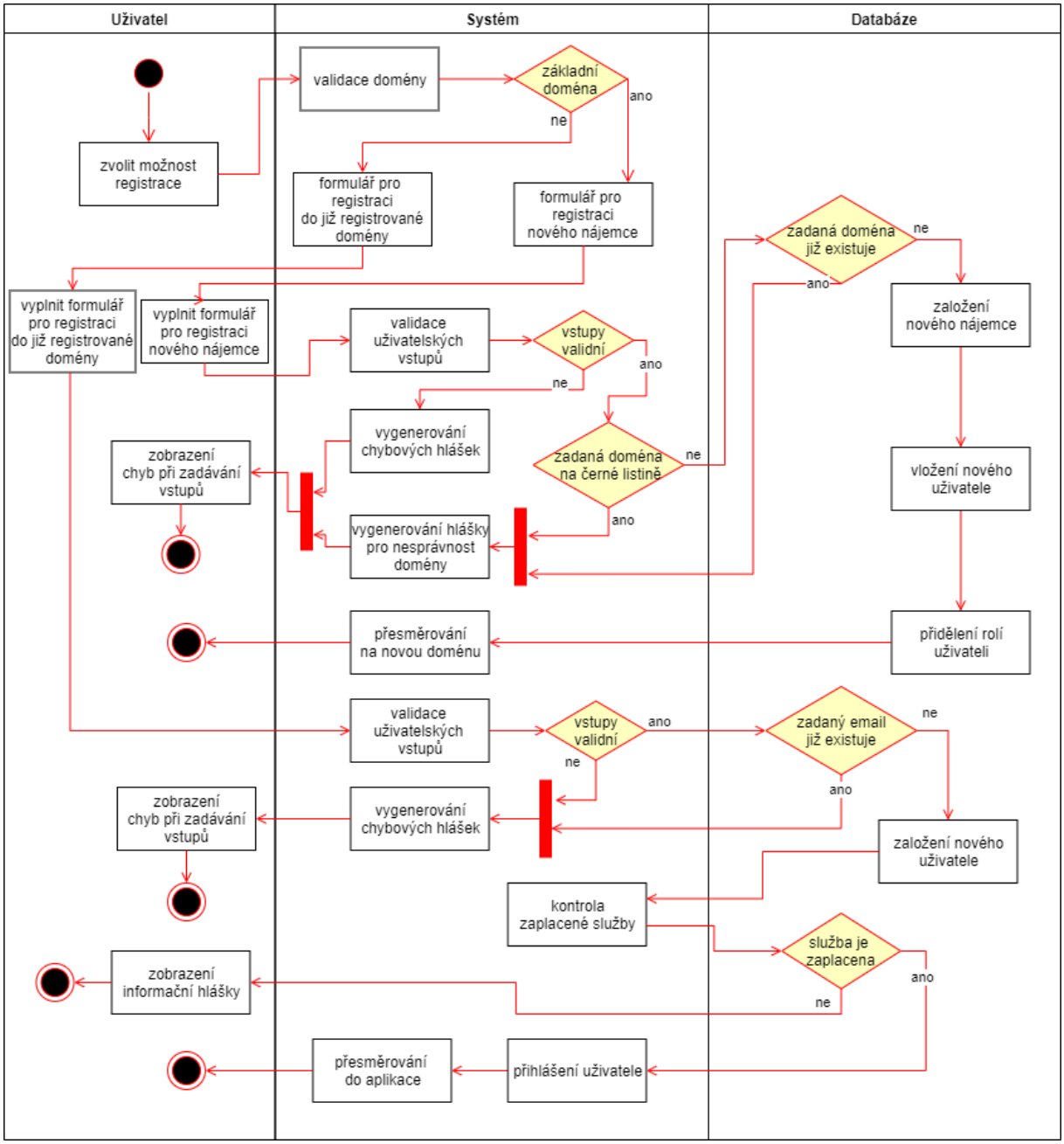
Tabulka 5: Scénář užití registrace pro nového nájemce

Primární aktér	Uživatel zakládající nového nájemce v systému
Hlavní scénář	1. Uživatel zvolí možnost registrace 2. Systém nabídne formulář pro registraci nového nájemce 3. Uživatel vyplní formulář a odešle jej 4. Systém validuje vstupy 5. Systém zkontroluje, že zadaná doména je unikátní a není obsažena v seznamu nepovolených doménových jmen 6. Systém spustí proceduru pro založení nového nájemce 7. Systém založí nového uživatele s rolí nájemce 8. Systém uživatele přesměruje na jeho novou doménu
Alternativní scénáře	A alternativní scénář pro bod 4 1 Systém zjistí, že uživatel špatně vyplnil vstupy 2 Systém vyznačí uživateli špatně zadané údaje a vypíše chybové hlášky B alternativní scénář pro bod 5 1 Systém zjistí, že jméno domény nemůže být použito 2 Systém vypíše uživateli hlášku, že doména již existuje

Tabulka 6: Scénář užití registrace do již registrované domény

Primární aktér	Uživatel registrující se na doménu nájemce
Hlavní scénář	1. Uživatel zvolí možnost registrace 2. Systém nabídne formulář pro registraci nového uživatele 3. Uživatel vyplní formulář a odešle jej 4. Systém ověří, že počet registrací nepřesáhl limit pro poskytovanou předplacenou službu 5. Systém validuje vstupy a ověří, že zadaný email již není registrován 6. Systém zaregistruje uživatele 7. Systém zkontroluje, že je poskytovaná služba řádně zaplacena 8. Systém přihlásí uživatele 9. Systém přesměruje uživatele na domovskou stránku aplikace
Alternativní scénáře	A alternativní scénář pro bod 4 1 Systém zjistí, že provozovaná služba nedovoluje další registraci 2 Systém vypíše uživateli chybovou hlášku B alternativní scénář pro bod 5 1 Systém zjistí, že uživatel vyplnil špatně formulář nebo že zadaný email je již registrován 2 Systém vyznačí uživateli špatně zadané údaje a vypíše chybové hlášky C alternativní scénář pro bod 7 1 Systém zjistí, že služba není zaplacena 2 Systém vrátí uživateli hlášku, že registrace proběhla, ale není možné se přihlásit

- **Diagram aktivit**



Obrázek 8: Diagram aktivit pro registraci

5 Porovnání s existujícími systémy

Na trhu v dnešní době existuje plno systémů, které jsou určeny ke správě úkolů a projektů. Zmínku zaslouží např. Trello[24], Asana[25], či česká alternativa Freelo[26]. Všechny zmíněné systémy jsou rozsáhlé a nabízí více funkcionalit, nežli systém implementovaný v této práci. Je třeba však vzít v potaz, že cílem této práce nebyla implementace systému pro správu úkolu a projektů, nýbrž demonstrace funkcionality SaaS modelu.

Funkcionalita implementovaného systému je velice podobná se základní funkcionalitou zmíněného IS Asana. Oba systémy jsou zaměřeny spíše na korporátní prostředí a na usnadnění řízení práce v týmu. V obou případech se také jedná o členění úkolů do projektů. Asana nabízí propracovanější interakci systému s uživatelem, kde si uživatel může rozčlenit úkoly do vlastních sekcí, či filtrovat podle různých parametrů: uživatel, který úkol zadal, datum, do kdy je třeba úkol splnit. Dále disponuje navíc možností zobrazení úkolů v kalendáři, přiřazení jednoho úkolu do více projektů, přidání přílohy k úkolům, nebo také umožňuje uživateli vytvořit projekt podle předem vytvořených šablon, které usnadní uživateli členění úkolů do jednotlivých sekcí. Oproti implementovanému systému, Asana neumí přímo přiřadit jeden úkol pro více uživatelů najednou, je třeba dodatečně zakládat podúkoly, které se teprve přiřadí ostatním. Nevýhodou systému Asana je chybějící možnost neplacené služby, kde má uživatel možnost využívat pouze omezené funkčnosti aplikace. Místo toho tato aplikace nabízí bezplatné zkušební období na 30 dní.

Druhý systém, kde byla porovnána funkcionalita, je výše zmíněný Trello. Základní funkcionalita je také velice obdobná, ovšem místo členění do projektů nabízí takzvané nástěnky, kde si následně uživatel může členit úkoly do sloupců. Oproti implementovanému systému, Trello neumožňuje zobrazení úkolů ze všech nástěnek najednou a uživatel při použití více nástěnek musí věnovat pozornost každé zvlášť. Tento systém se zdá méně přehledný a méně přívětivý pro uživatele, než u implementovaného systému. Trello disponuje nástrojem Butler, který může uživateli sloužit pro automatizaci rozdělení úkolů do sloupců nebo také přizpůsobit pracovní prostředí uživatele, včetně přidání tlačítek s nastavitelnou funkčností.

Jak bylo zmíněno výše, oba systémy jsou rozsáhlé a vyvíjené dlouhou dobu s daleko větší kapacitou lidských zdrojů. Je proto velice nevyvážené porovnávat implementovaný systém s výše zmíněnými. Ovšem v implementovaném systému je možnost uživatelského nastavení, kterou uvedené systémy nedisponují. Aplikace tohoto nastavení umožňuje zobrazení úkolů jen z vybraných projektů z dostupného seznamu projektů. Rovněž implementovaný systém dokáže uživatele vizuálně upozornit na úkoly blížící se termínu splnění.

6 Implementace

6.1 Registrace

Nejrozsáhlejší funkcí v implementovaném systému je registrace. Tato funkce se stará jak o správné vykonání registrace uživatele, tak o vykonání registrace nového nájemce. Ve výpisu 1 jsou uvedeny podmínky funkce, které slouží ke kontrole, zda je možné operace provést.

Řádek 2 udává validaci uživatelských vstupů. Mimo implicitní validaci modelu je potřeba explicitně validovat doménu a jméno nájemce, které nedisponují atributem hlídajícím zadanou hodnotu, protože při registraci do již zavedené domény zůstávají bez hodnoty. Vzhledem k tomu, že je název domény součástí URL, je potřeba provést validaci, kdy dochází ke kontrole, zda zvolená doména je složena pouze z písmen, viz řádek 3. Funkce dále validuje existenci domény a kontroluje, zda zvolený název domény není přítomen na seznamu nepovolených doménových jmen, tak aby nedošlo k možnému konfliktu jmen na úrovni databáze, viz řádek 5.

V „else“ větvi ve výpisu 1 validuje případy pro registraci uživatele do domény nájemce. Řádek 8 ukazuje příklad validace pro jednotlivé stupně předplácených služeb a jeho účel je omezit počet registrací v doméně. Poslední podmínka, zobrazená na řádku 11, kontroluje pomocí autorizační kontroly (kap. 6.2), zda má doména řádně zaplacenou službu.

Princip funkce jako celku je schématicky znázorněn v dříve uvedeném scénáři a diagramu aktivit v kapitole 4.4.3.

```
1  if (HttpContext.Items["domain"] as string == "default"){
2  //kontrola vstupů při zakládání nového nájemce
3  if (!string.IsNullOrEmpty(signUp.TenantDomain) && !string.IsNullOrEmpty(signUp.
       TenantName) && ModelState.IsValid)
4     if (Regex.Matches(signUp.TenantDomain, @"[a-zA-Z]").Count == signUp.
          TenantDomain.Length)
5        var blacklist = new string[] { "default", "admin", "NewTenantUser", "sa",
              "helper", "dbo", "guest", "sys", "ttask" };
6        if (!_tt.TenantAlreadyExists(signUp.TenantDomain) && !blacklist.Contains(
             signUp.TenantDomain))
7
8  else{
9  //kontrola vstupů při registraci uživatele
10 if ((idService == (int)Services.Basic && noUsers < (int)Basic.NoUsers) || (
       idService == (int)Services.Pro && noUsers < (int)Pro.NoUsers) || idService
       == (int)Services.Business)
11    if (ModelState.IsValid)
```

```
12      var TenantPolicyResult = await _authorization.AuthorizeAsync(User, "
            TenantPolicy");
13      if (TenantPolicyResult.Succeeded)
```

Výpis 1: Podmínky ve funkci registrace

6.2 Autorizační kontrola

Pro zajištění správného chodu aplikace bylo zapotřebí implementovat autorizační kontrolu, která bude hlídat správnost domény, přihlášeného uživatele a zaplacení služby. Kontrola je zobrazena ve výpisu 2. Při každém HTTP požadavku je potřeba validovat doménu, jako provádí funkce volaná na řádku 3. Funkce dále kontroluje, zda přihlášený uživatel skutečně spadá pod danou doménu. Tato realita by mohla nastat při shodě uživatelských jmen pro rozdílné domény a je ošetřena pomocí pomocné cookie zavedené při přihlášení uživatele (viz řádek 5). Dále se tato autorizační kontrola stará o zakázání přístupu do domény v případě, že není zaplacená, čehož je docíleno pomocí podmínky na řádku 12. Tato podmínka při nezaplacené doméně dovoluje přístup pouze, když HTTP požadavek směřuje z profilu, což umožní přístup pouze nájemci k zaplacení nebo změně služby.

```
1  protected override Task HandleRequirementAsync(AuthorizationHandlerContext
       context, TenantPolicyRequirement requirement)
2  {
3      if (CorrectDomain())
4      {
5          if (_httpContext.Request.Cookies["Identity.Domain"] == _httpContext.
               Items["domain"] as string || _httpContext.Request.Path.Value.
               Contains("SignIn") || _httpContext.Request.Path.Value.Contains("
               SignUp"))
6          {
7              var serviceOrder = _serviceOrderTable.
                   GetNewestServiceOrderToTenantByTenantId(_tenantTable.GetTenantId
                   (_httpContext.Items["domain"] as string));
8              if (serviceOrder.IdService == (int)Services.Basic)
9                  context.Succeed(requirement);
10             else if (serviceOrder.IdPaymentNavigation.Paid.HasValue)
11             {
12                 if ((serviceOrder.IdPaymentNavigation.Price == serviceOrder.
                       IdServiceNavigation.Price && serviceOrder.IdPaymentNavigation
                       .Paid.Value.AddDays((int)Payment.Duration) > DateTime.Now) ||
                        _httpContext.Request.Path.Value.Contains("Profile"))
```

```
13                  context.Succeed(requirement);
14              }
15          }
16      }
17      return Task.CompletedTask;
18  }
19
20  public bool CorrectDomain()
21  {
22      var tmp = _httpContext.Items["domain"] as string;
23      var domainList = _tenantTable.GetAllDomains();
24      return (domainList.Contains(tmp) && tmp != "default")
25  }
```

Výpis 2: Autorizační kontrola

6.3 Izolace dat

Mezi nejnáročnější procesy při vývoji patřil proces návrhu a implementace izolace dat. Část procedury řešící tuto problematiku je uvedena ve výpisu 3. Tato procedura je spouštěna při založení nového nájemce. Vstupní parametry procedury uvedené ve výpisu jsou *p_domain* a *p_password*. V parametru *p_domain* je obsažen název domény, který slouží jako identifikátor pro nájemce. Tento identifikátor je použitý jako uživatelské jméno pro přihlášení na SQL server, jméno uživatele v databázi a jméno schématu náležícího nájemci. Parametr *p_password* obsahuje hash pro danou doménu, který je řešen v aplikační logice pomocí algoritmu SHA256. Procedura nejprve vytvoří login na straně SQL serveru a uživatele na straně databáze (viz řádky 1-5). Dále vytvoří separátní schéma, kde je nově vytvořený uživatel nastaven majitelem, jak je znázorněno na řádcích 7-9. Na řádku 15 je uživatel přiřazen do role *tenants*, která byla vytvořena předem a povoluje uživateli přístup do sdíleného schématu *dbo* a zakazuje přístup k systémovým schématům *sys* a *INFORMATION_SCHEMA* (viz řádky 18-20). Díky těmto přístupovým pravidlům je dosaženo izolace a logického oddělení dat jednotlivých nájemců, kdy je nájemci povoleno přistupovat pouze k datům svého a sdíleného schématu.

```
1  SET @sqlUser = 'USE ttask;
2  CREATE LOGIN ' + @p_domain +
3      ' WITH PASSWORD = ''' + @p_password + ''', DEFAULT_DATABASE=ttask,
         DEFAULT_LANGUAGE=[English], CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF;
4  CREATE USER ' + @p_domain + ' FROM LOGIN ' + @p_domain + ';'
5  EXEC (@sqlUser)
6
```

```
7  SET @sqlSchema = 'CREATE SCHEMA ' + @p_domain +
8     ' AUTHORIZATION ' + @p_domain + ';'
9  EXEC (@sqlSchema)
10
11 SET @sqlDef = 'ALTER USER ' + @p_domain +
12    ' WITH DEFAULT_SCHEMA = ' + @p_domain + ';'
13 EXEC (@sqlDef)
14
15 EXEC sp_addrolemember 'tenants', @p_domain;
16
17 /* Při zakládání databáze byla roli "tenants" přidělena tato přístupová
      pravidla: */
18 GRANT SELECT, UPDATE, DELETE, INSERT ON SCHEMA::dbo TO [tenants];
19 DENY SELECT, UPDATE, DELETE, INSERT ON SCHEMA::sys TO [tenants];
20 DENY SELECT, UPDATE, DELETE, INSERT ON SCHEMA::INFORMATION_SCHEMA TO [tenants];
```

Výpis 3: Část procedury založení nového nájemce

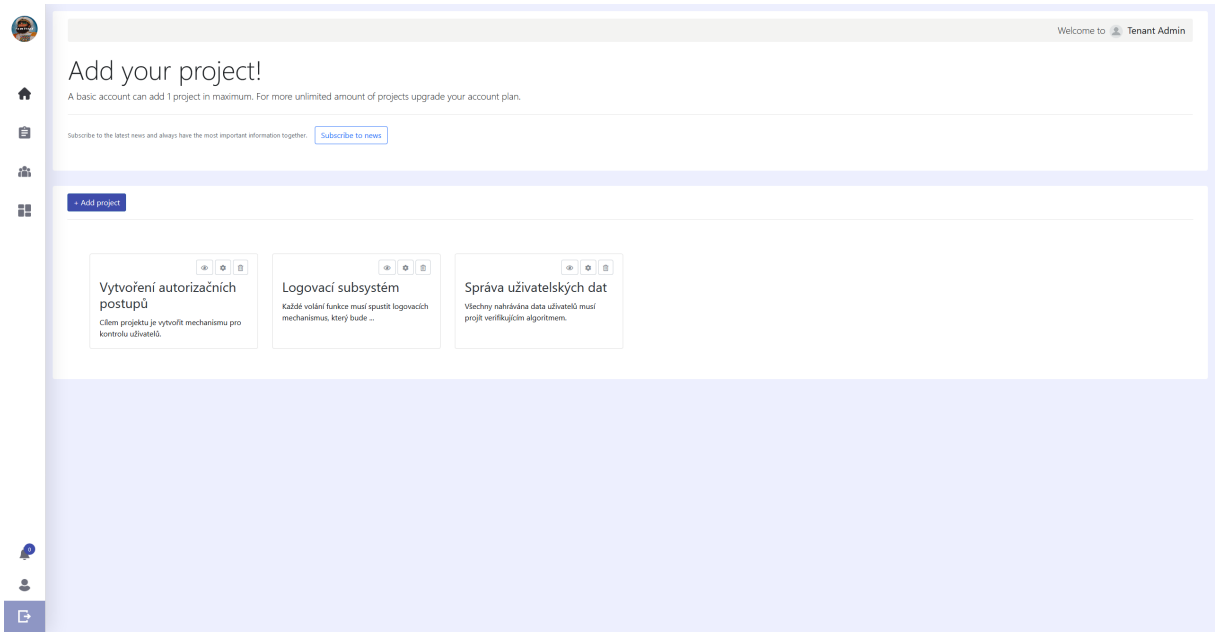
Pro větší bezpečnost existují v databázi další dva uživatelské účty - *default* a *NewTenantUser*. Uživatel *default* má přístupová práva pouze ke sdílenému schématu *dbo* a využívá se k vykonávání metod využívajících SQL dotazů nad sdílenými tabulkami, kdy není potřeba přistupovat k datům jednotlivých nájemců. Uživatel *NewTenantUser* je speciálně vytvořený pro volání procedury registrování nového nájemce, která vyžaduje zvýšená přístupová práva. Tento uživatel má v rámci SQL serveru roli *securityadmin* a v rámci databáze roli *db_owner*, které jsou nutné pro vykonání procedury.

Díky implementačnímu postupu sdílené databáze a izolovaného schématu není třeba implementovat složitou logiku na aplikační vrstvě, která řeší pouze způsob připojení k databázi. Výpis 4 zobrazuje části zdrojového kódu věnující se této problematice. Aby byl zajištěn přístup ke konkrétním datům odpovídajícího nájemce, byl implementován middleware (řádky 2-4), který při každém HTTP požadavku uloží název domény (identifikátor nájemce) do položky HTTP kontextu. Jak zobrazují řádky 7 a 8, díky této položce, kterou využívá aplikační databázový kontext, se zkonstruuje připojovací řetězec pro spojení s databází. V případě, kdy se přistupuje ke sdíleným tabulkám databáze nebo se volá procedura pro registraci nového nájemce, se využívá sdílený databázový kontext (viz řádky 11-14). V případě registrace nového nájemce se využívá připojení přes databázového uživatele *NewTenantuser*, čehož je docíleno funkcí využívající návrhový vzor Factory za použití principu Inversion Of Control, která je zobrazena na řádku 20. V ostatních případech je připojení k databázi provedeno pomocí databázového uživatele *default*. [27]

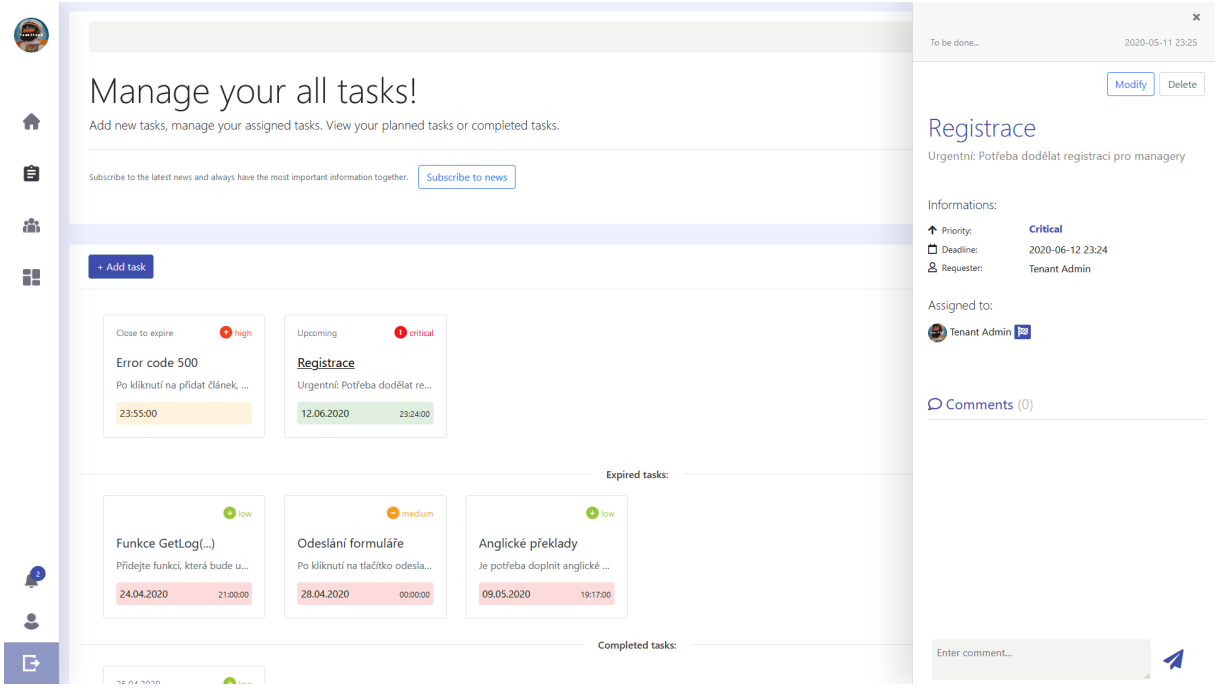
```
1  //Ukázka z Middlewaru pro zjištění domény
2  string url = context.Request.Path.Value;
3  var domain = url.Split('/')[1] == string.Empty ? "default" : url.Split('/')[1];
4  context.Items["domain"] = domain;
5
6  //Ukázka konstrukce připojovacího řetězce v aplikačním databázovém kontextu
7  string domain = _httpContext.Items["domain"] as string;
8  var connectionString = Configuration.GetConnectionString("TenantConnection").
       Replace(":domainPass", Domain.GetDomainHash(domain)).Replace(":domain",
       domain);
9
10 //Ukázka vytvoření spojení přes sdílený databázový kontext
11 if (!_isNewTenant)
12    optionsBuilder.UseSqlServer(Configuration.GetConnectionString("
          DefaultConnection"));
13 else
14    optionsBuilder.UseSqlServer(Configuration.GetConnectionString("
          NewTenantConnection"));
15
16 //Ukázka volání sdíleného databázového kontextu pro registraci nového nájemce
       za použití Inversion of Control
17 public NewTenantProcedure(Func<bool, SharedDbContext> dbContextFunction) { _db
       = dbContextFunction(true); }
18
19 //Ukázka využití návrhového vzoru Factory
20 services.AddScoped<Func<bool, SharedDbContext>>((provider) => (isNewTenantCon
       => new SharedDbContext(isNewTenantCon, provider.GetService<IConfiguration
       >())));
```

Výpis 4: Připojení k databázi v aplikační logice

7 Grafické rozhraní



Obrázek 9: Grafické rozhraní domovské stránky za využití UI/UX principu [28]



Obrázek 10: Grafické rozhraní seznamu úkolů za využití UI/UX principu [28]

8 Závěr

V rámci této bakalářské práce bylo jedním z cílů seznámení se s modelem SaaS. V úvodu práce byl model SaaS popsán a byly popsány jeho klady a zápory. Dále byla provedena analýza implementačních postupů a architektur, na kterých je model SaaS založen. Architektury single-tenant a multi-tenant jsem následně popsal a uvedl jejich výhody a nevýhody. Následně jsem představil tři hlavní implementační postupy multi-tenant architektury. Na závěr teoretické části jsem výše popsané architektury porovnal.

Dalším cílem této práce bylo implementovat IS demonstrující model SaaS, kterému se věnovala praktická část práce. S implementací modelu SaaS jsem neměl žádné předchozí zkušenosti a byla pro mě neznámou. Tématem IS pro otestování modelu SaaS byl zvolen IS pro správu úkolů a projektů. Jako implementační řešení SaaS modelu byla zvolena multi-tenant architektura. Tato volba byla provedena v zásadě kvůli snadnější možnosti testování a daleko menší infrastruktuře serverů. Dalším faktorem, který vedl k této volbě, byla daleko snazší správa systémových prostředků a nájemců.

V úvodu praktické části jsem představil implementovaný systém a popsal využité technologie při implementaci. S technologiemi, jako jsou ASP.NET Core Identity [29] a Entity Framework Core [20], jsem se setkal poprvé a dalo se očekávat, že se jejich implementace neobejde bez obtíží. Problém nastal při využívání ASP.NET Core Identity pro správu uživatelů, kde jsem si musel pomoct explicitními metodami k docílení správného fungování multi-tenant prostředí. Následovala datová a funkční analýza implementovaného systému. V datové analýze jsem znázornil ERD modely databáze. Následně jsem popsal důvody zvolených implementačních postupů: sdílená databáze, izolované schéma, izolace dat, optimalizace a výkon systému. Ve funkční analýze jsem definoval veškeré uživatelské role systému a popsal jeho stěžejní funkce. Nejrozsáhlejší funkci jsem definoval ve formě scénářů a diagramu aktivit. Po funkční analýze následovalo porovnání s existujícími systémy s podobnou funkcionalitou, kde jsem implementovaný systém porovnal s často využívanými systémy Trello a Asana. V rámci kapitoly Implementace jsem popsal tři funkcionality systému: registraci do systému, autorizační kontrolu a izolaci dat. Při implementaci jsem řešil algoritmický problém izolace dat, kdy ke zdárnému řešení vedlo nespočet pokusů a nápomocná online dokumentace. Závěrem práce jsem znázornil grafické rozhraní implementovaného systému. V rámci implementace byl kladen velký důraz na izolaci dat, kde bylo docíleno požadovaného výsledku.

Implementovaný IS však není bezchybný a perfektně optimalizovaný. Co se týče optimalizace, dalo by se jistě omezit počet spojení systému s databází přesunutím části logiky na stranu databáze, např. ve formě uložených procedur. Rozšířit by se dala také funkcionalita systému například o komunikaci s komponentami třetích stran, což by mohlo umožnit uživatelům integraci do již používaných aplikací. Dále by se dal systém rozšířit z pohledu uživatelského rozhraní, které by obsahovalo moderní trendy v integraci na mobilní zařízení.

Ověřil jsem si, že implementace IS je náročný proces, který vyžaduje široké znalosti. Často

také vyžaduje více lidí, kteří se na něm podílí, kde každý má svou podstatnou roli. Pro následování moderních trendů je také důležitý neustálý vývoj systému.

Literatura

1. *Microsoft Office 365* [online] [cit. 2020-04-04]. Dostupné z: https://www.office.com/.
2. *Google Apps* [online] [cit. 2020-04-04]. Dostupné z: https://gsuite.google.cz/intl/cs/.
3. *Co je SaaS?* [online] [cit. 2020-04-05]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-is-saas/.
4. DURČÁK, Pavel. *Software jako služba* [online]. 2017-11-14 [cit. 2020-04-05]. Dostupné z: https://www.napocitaci.cz/33/software-jako-sluzba-uniqueidgOkE4NvrWuNY54vrLeM675EOGV3 9l4jAfKtmfKBkk/.
5. TSYGANOV, Dmitry. *FUNDAMENTAL PROPERTIES OF SAAS ARCHITECTUREE: LITERATURE REVIEW AND ANALYSIS OF DEVELOPMENT PRACTICES*. Lappeenranta, Moscow, 2018. Master's Thesis. Lappeenranta University of Technology.
6. *Software jako služba: Nebe, peklo, ráj* [online]. 2015-06-08 [cit. 2020-04-06]. Dostupné z: http://www.businessit.cz/cz/software-jako-sluzba-nebe-peklo-raj.php.
7. ROUSE, Margaret. *single-tenancy* [online]. 2020-01 [cit. 2020-04-11]. Dostupné z: https://searchcloudcomputing.techtarget.com/definition/single-tenancy.
8. RAI, RAJESH. *What is Multi-tenant SaaS Application?* [online] [cit. 2020-04-12]. Dostupné z: https://mrrajeshrai.com/what-is-multi-tenant-saas-application.
9. WATTS, Stephen. *What's The Difference Between Single Tenant and Multi-Tenant?* [online]. 2018-06-19 [cit. 2020-04-11]. Dostupné z: https://www.bmc.com/blogs/single-tenant-vs-multi-tenant/.
10. LOW, Terry. *Single-Tenant Vs Multi-Tenant Hosting* [online]. 2018-04-09 [cit. 2020-04-11]. Dostupné z: https://shareplm.com/single-tenant-vs-multi-tenant-hosting/.
11. OOSTHUIZEN, Aubrey D. *Developing a multi-tenant, media-marketplace architecture with Microsoft Azure and ASP.NET*. Brno, 2014. Master's Thesis. MASARYKOVA UNIVERZITA FAKULTA INFORMATIKY.
12. *A beginner's guide to database multitenancy* [online]. 2019-08-14 [cit. 2020-04-17]. Dostupné z: https://vladmihalcea.com/database-multitenancy/.
13. *Multi-tenant SaaS database tenancy patterns* [online]. 2019-01-25 [cit. 2020-04-17]. Dostupné z: https://docs.microsoft.com/en-us/azure/sql-database/saas-tenancy-app-design-patterns.
14. *ASP.NET Core* [online] [cit. 2020-04-25]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.1.
15. *JavaScript* [online] [cit. 2020-04-25]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

16. *AJAX* [online] [cit. 2020-04-25]. Dostupné z: https://www.w3schools.com/xml/ajax_intro.asp.

17. *Bootstrap* [online] [cit. 2020-04-25]. Dostupné z: https://getbootstrap.com/.

18. *CSS* [online] [cit. 2020-04-25]. Dostupné z: https://www.w3.org/Style/CSS/Overview.en.html.

19. *SQL Server 2019* [online] [cit. 2020-04-25]. Dostupné z: https://www.microsoft.com/en-us/sql-server/sql-server-2019.

20. *Entity Framework Core* [online] [cit. 2020-04-25]. Dostupné z: https://docs.microsoft.com/en-us/ef/core/.

21. *IIS* [online] [cit. 2020-04-25]. Dostupné z: https://www.iis.net/.

22. *Windows Server 2019* [online] [cit. 2020-04-25]. Dostupné z: https://www.microsoft.com/en-us/cloud-platform/windows-server.

23. *IIS Express* [online] [cit. 2020-04-25]. Dostupné z: https://docs.microsoft.com/en-us/iis/extensions/introduction-to-iis-express/iis-express-overview.

24. *Trello* [online] [cit. 2020-05-02]. Dostupné z: https://trello.com/.

25. *Asana* [online] [cit. 2020-05-02]. Dostupné z: https://app.asana.com/.

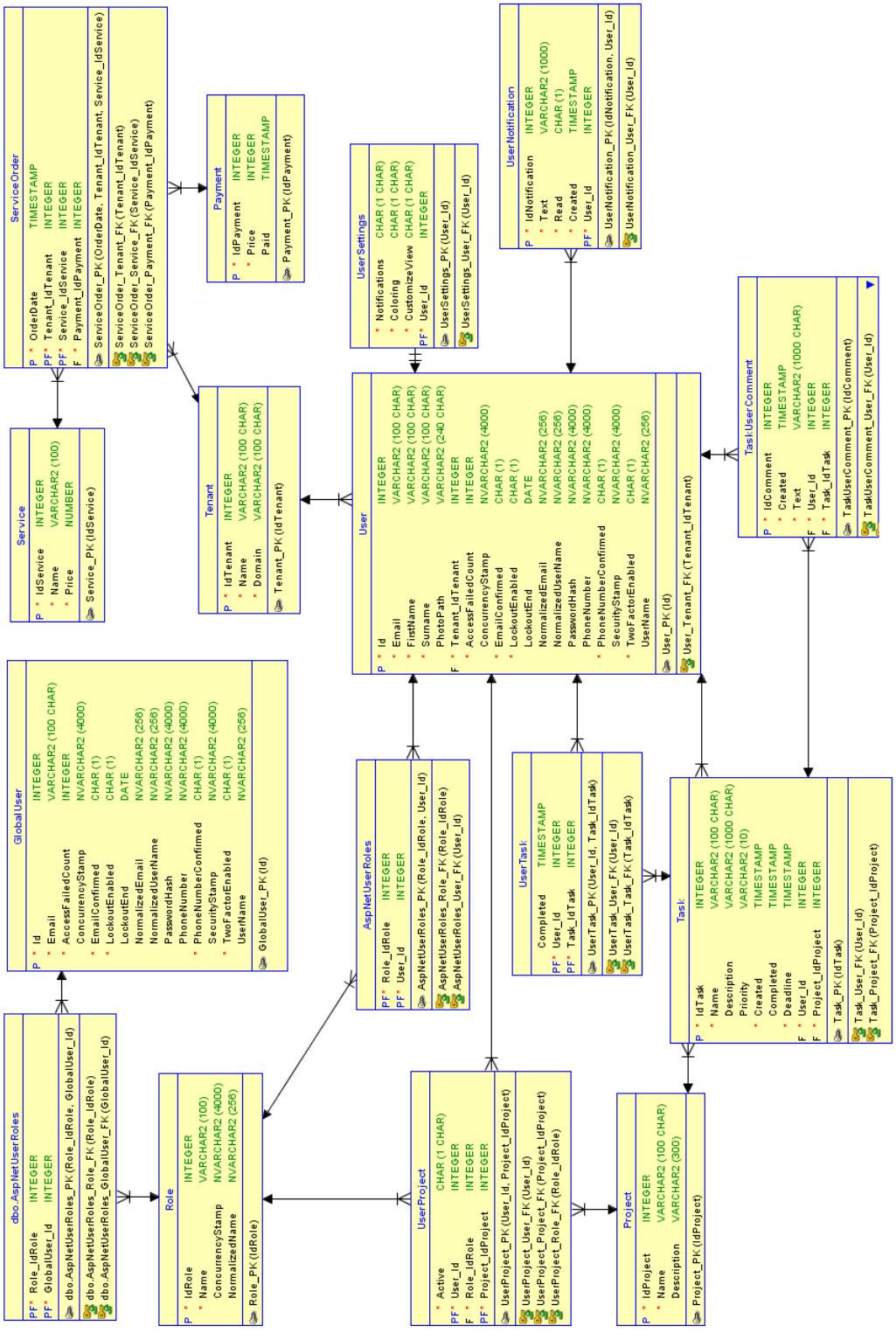
26. *Freelo* [online] [cit. 2020-05-02]. Dostupné z: https://www.freelo.cz/cs.

27. FREEMAN, Adam. *Pro ASP.NET Core MVC*. 6th ed. Apress, 2016. ISBN 978-1484203989.

28. CABRERA, James. *Modular Design Frameworks: a Project-based Guide for UI/UX Designers*. Apress, 2017. ISBN 978-1484216873.

29. *ASP.NET Core Identity* [online] [cit. 2020-05-08]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-3.1&tabs=visual-studio.

A Relační model databáze



Obrázek 11: Relační model databáze

B Uživatelská dokumentace

Uživatelská dokumentace pro implementovanou aplikaci je dostupná na adrese `https://bit.ly/2zxVV1v`.

C Elektronická příloha

Elektronická příloha obsahující:

- Zdrojový kód implementovaného systému
- SQL script pro vytvoření databáze a potřebných loginů, uživatelů, schémat, tabulek, procedur a testovacích dat
- SQL script pro odstranění databáze a využitých loginů